

Date=23/12/2020

Lecture By= Akash Handa

Subject ⇒ Hooks In react

IN PREVIOUS LECTURE (QUICK RECAP) Date-22/12/2020	In Today's Lecture (Overview)
applyMiddleware(...middleware) Arguments Tips# Redux Promise	Introducing Hooks When to use a Hooks Rules of Hooks 1. Only call Hooks at the top level 2. Only call Hooks from React functions Prerequisites for React Hooks Hooks State Hooks Effect Effects without Cleanup Effects with Cleanup Custom Hooks

Introducing Hooks

Hooks are a new addition in React 16.8. They let you use state and other React features without writing a class.

```
import React, { useState } from 'react';

function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);

  return (
```

```
<div>
  <p>You clicked {count} times</p>
  <button onClick={() => setCount(count + 1)}>
    Click me
  </button>
</div>
);
}
```

This new function `useState` is the first “Hook” we’ll learn about, but this example is just a teaser. Don’t worry if it doesn’t make sense yet!

You can start learning Hooks on the next page. On this page, we’ll continue by explaining why we’re adding Hooks to React and how they can help you write great applications.

When to use a Hooks

If you write a function component, and then you want to add some state to it, previously you do this by converting it to a class. But, now you can do it by using a Hook inside the existing function component.

Rules of Hooks

Hooks are similar to JavaScript functions, but you need to follow these two rules when using them. Hooks rule ensures that all the stateful logic in a component is visible in its source code. These rules are:

1. Only call Hooks at the top level

Do not call Hooks inside loops, conditions, or nested functions. Hooks should always be used at the top level of the React functions. This rule ensures that Hooks are called in the same order each time a components renders.

2. Only call Hooks from React functions

You cannot call Hooks from regular JavaScript functions. Instead, you can call Hooks from React function components. Hooks can also be called from custom Hooks.

Pre-requisites for React Hooks

1. Node version 6 or above
2. NPM version 5.2 or above
3. Create-react-app tool for running the React App

Hooks State

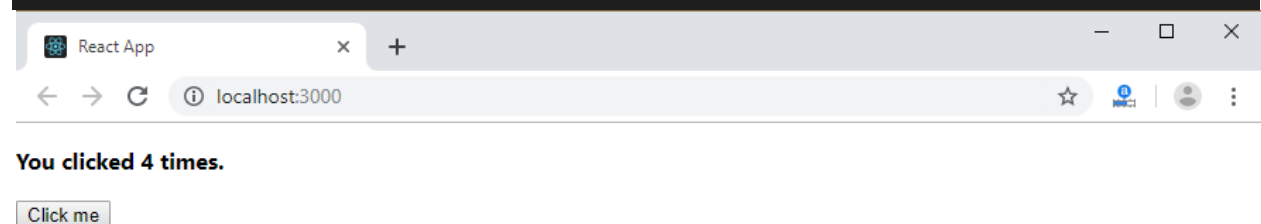
Hook state is the new way of declaring a state in React app. Hook uses `useState()` functional component for setting and retrieving state. Let us understand Hook state with the following example.

```
import React, { useState } from 'react';

function CountApp() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}

export default CountApp;
```



The screenshot shows a web browser window with the title 'React App' and the address bar displaying 'localhost:3000'. The page content shows the text 'You clicked 4 times.' followed by a button labeled 'Click me'.

In the above example, `useState` is the Hook which needs to call inside a function component to add some local state to it. The `useState` returns a pair where the first element is the current state value/initial value, and the second one is a function which allows us to update it. After that, we will call this function from an event handler or somewhere else. The `useState` is similar to `this.setState` in class. The equivalent code without Hooks looks like as below.

App.js

```
import React, { useState } from 'react';

class CountApp extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0
    };
  }
  render() {
    return (
      <div>
        <p><b>You clicked {this.state.count} times</b></p>
        <button onClick={() => this.setState({ count: this.state.count + 1
      })}>
          Click me
        </button>
      </div>
    );
  }
}

export default CountApp;
```

Hooks Effect

The Effect Hook allows us to perform side effects (an action) in the function components. It does not use components lifecycle methods which are available in class components. In other words, Effects Hooks are equivalent to `componentDidMount()`, `componentDidUpdate()`, and `componentWillUnmount()` lifecycle methods.

Side effects have common features which the most web applications need to perform, such as:

- Updating the DOM,

- Fetching and consuming data from a server API,
- Setting up a subscription, etc.

Let us understand Hook Effect with the following example.

```
import React, { useState, useEffect } from 'react';

function CounterExample() {
  const [count, setCount] = useState(0);

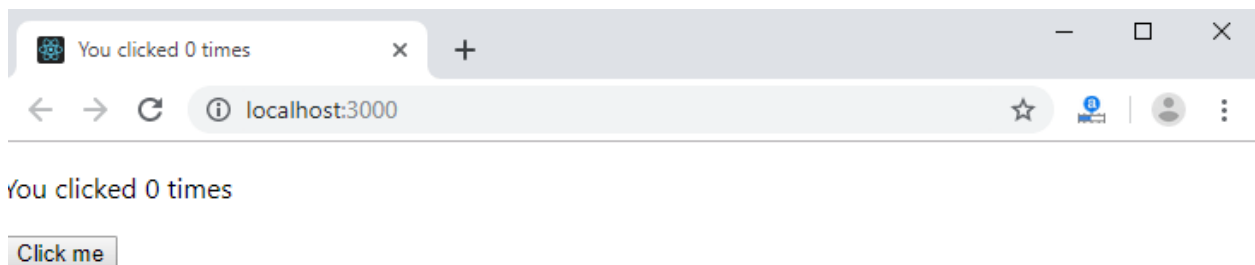
  // Similar to componentDidMount and componentDidUpdate:
  useEffect(() => {
    // Update the document title using the browser API
    document.title = `You clicked ${count} times`;
  });

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}

export default CounterExample;
```

The above code is based on the previous example with a new feature which we set the document title to a custom message, including the number of clicks.

Output:



In React component, there are two types of side effects:

1. Effects Without Cleanup
2. Effects With Cleanup

Effects without Cleanup

It is used in `useEffect` which does not block the browser from updating the screen. It makes the app more responsive. The most common example of effects which don't require a cleanup are manual DOM mutations, Network requests, Logging, etc.

Effects with Cleanup

Some effects require cleanup after DOM updation. For example, if we want to set up a subscription to some external data source, it is important to clean up memory so that we don't introduce a memory leak. React performs the cleanup of memory when the component unmounts. However, as we know that, effects run for every render method and not just once. Therefore, React also cleans up effects from the previous render before running the effects next time.

Custom Hooks

A custom Hook is a JavaScript function. The name of custom Hook starts with "use" which can call other Hooks. A custom Hook is just like a regular function, and the word "use" in the beginning tells that this function follows the rules of Hooks. Building custom Hooks allows you to extract component logic into reusable functions.